PROMON

How to attack (and secure) an Android app: An introduction

droidconberlin

Benjamin Adolphi <benjamin@promon.de>

whoami

- Head of the security research team at Promon.
- Breaking and securing apps since 2011.
- Passionate reverse engineer.
- Terrible at making apps.

Introduction

- What is this workshop about?
  - Showing the view of an attacker.
  - Mostly practical demonstrations.
  - Discussion of countermeasures.
- Material: https://github.com/badolphi/droidcon-berlin

Reverse engineering

- Understanding how an app works.
- Reveal secrets in it.
- First step of an attacker.
- Two complementary approaches: Static and dynamic
- On Android
  - Java code (Java, Kotlin)
  - Native code (C, C++, Dart, …)

Reverse engineering Java code

- Code in classes.dex file(s).
- Dalvik bytecode executed in VM.
- Requires disassembler[1] or decompiler[2].

[1] https://github.com/iBotPeaches/Apktool

[2] https://github.com/skylot/jadx

Reverse engineering native code

- Code is found in .so files.
- Executed directly on the CPU.
- There are many good disassemblers/decompilers [1,2,3,4].

BINARYNINJA

GHIDRA

[1] https://hex-rays.com/ida-pro

[2] https://binary.ninja

[3] https://github.com/NationalSecurityAgency/ghidra

[4] https://rada.re

PROMON

Demo

Protecting against reverse engineering

- Impossible to prevent.
- Obfuscation can make it harder.
- Some things you can do
  - Rename/remove identifiers.
  - Encrypt strings.
  - Use reflection.
  - Use native code.
- Ideally done with a tool[1,2,3,4,5].

[1] https://r8.googlesource.com/r8

[2] https://www.guardsquare.com/proguard

[3] https://github.com/obfuscator-llvm/obfuscator

[4] https://obfuscator.re/omvll

[5] https://obfuscator.re/dprotect

Repackaging

- Modifying app on disk.
- Change code to change behavior.
- Change resources to change look.

Patching Java code

- Modify classes.dex file(s).
- Direct binary patching can be tricky.
- Tools like apktool make this easy
  - Disassemble to smali.
  - Modify smali.
  - Re-assemble to apk.

Patching Native code

- Modify .so file(s).
- Can be done manually.
- Disassemblers/decompilers usually make this easier.
- Requires available space.

Demo

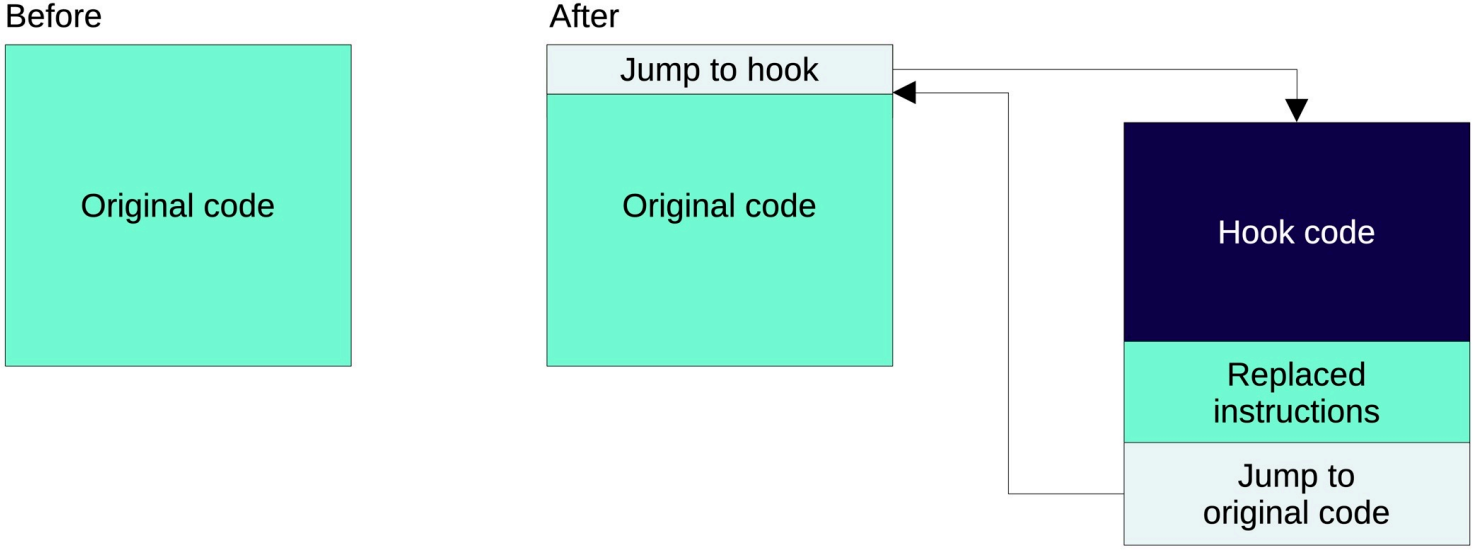
Protecting against Repackaging

- Implement anti-tampering mechanisms
  - Check APK signature and signer.
  - Implement checksumming mechanism.
- Can also be patched.
- Obfuscation can make this more difficult.
- Multiple independent mechanisms can make this more difficult.

Hooking

- Modify the app while it runs.
- Change code to change behavior.
- Useful for dynamic reverse engineering.

How hooking works



PROMON

Hooking Java code

- Code is executed in VM.
- Could be compiled ahead of time or just in time.
- Requires modifying the VM.
- Popular hooking frameworks
  - LSPosed[1]
  - Frida[2]

FRIDA

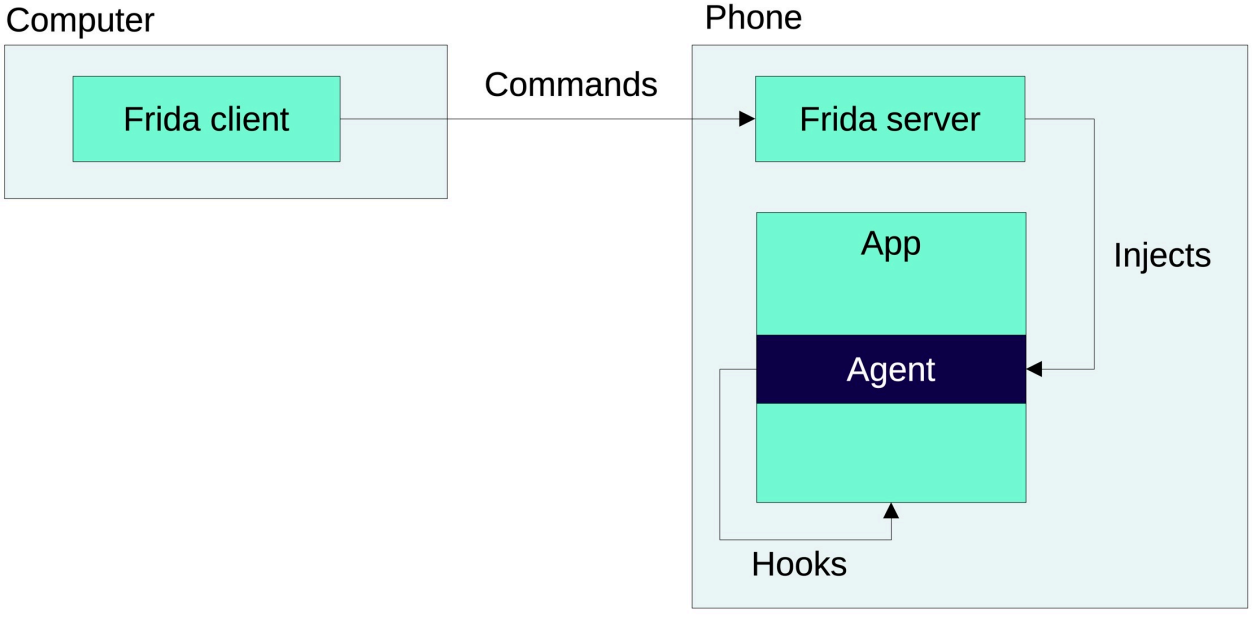
[1] https://github.com/LSPosed/LSPosed

[2] https://frida.re

Hooking native code

- Overwrite code in memory.
- Not completely trivial.
- Frida is a popular framework to use.

How Frida works in our use case



PROMON

Demo

Protecting against Hooking

- Detect hooks
  - Check for code modifications in memory.
- Detect hooking framework
  - Check for suspicious files, libraries and communication channels.
- Can also be hooked.
- Obfuscation and multiple independent mechanisms make it harder.

Debugging

- Inspect app while it is running.
- Useful for dynamic reverse engineering.
- Change state to change behavior.

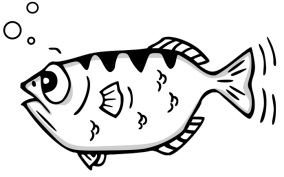
Debugging Java code

- Usually requires `android:debuggable`.
- Options
  - Patch manifest, e.g. with ManifestEditor[1].
  - Set `ro.debuggable`.
  - Hooking Zygote or ART.
- Use Android Studio to debug APK.

[1] https://github.com/WindySha/ManifestEditor

Debugging native code

- Launch debug server on Android device.
- Connect from client on computer.
- Usually requires root.
- gdb[1] and lldb[2] are popular.



[1] https://www.sourceware.org/gdb

[2] https://lldb.llvm.org

PROMON

Demo

Protecting against Debugging

- Some things you can do against Java debuggers
  - Check manifest.
  - Check `ro.debuggable`.
  - Use the `Debug.isDebuggerConnected`.
- Some things you can do against native debuggers
  - Check `/proc/self/status`.
  - Check for root access.
- Can all be bypassed by debugger.
- Obfuscation and multiple independent mechanisms make it harder.

Summary

- Is this a problem for you?
- Possible to implement countermeasures yourself.
- Better than doing nothing but probably not too effective.
- It might be worth considering getting help.

PROMON

Thank you!

Benjamin Adolphi
Security Ninja
benjamin@promon.de